Blackbird: Object-Oriented Planning, Simulation, and Sequencing Framework Used by Multiple Missions

Christopher R. Lawler, Forrest L. Ridenhour, Shaheer A. Khan, Nicholas M. Rossomando, Ansel Rothstein-Dowden
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
{christopher.r.lawler, forrest.ridenhour, shaheer.khan, nicholas.m.rossomando, ansel.rothstein-dowden}@jpl.nasa.gov

Abstract— **Every JPL flight mission relies on activity planning and sequence generation software to perform operations. Most such tools in use at JPL and elsewhere use attribute-based schemas or domain-specific languages (DSLs) to define activities. This reliance poses user training, software maintenance, performance, and other challenges. To solve this problem for future missions, a new software called Blackbird was developed which allows engineers to specify behavior in standard Java. The new code base has over an order of magnitude fewer lines of code than other JPL planning software, since no DSL or schema interpreter is needed. The use of Java for defining activities also allows mission adapters to debug their code in an integrated development environment, seamlessly call external libraries, and set up truly multi-mission models. These efficiency gains have significantly reduced the amount of development effort required to support the software. This paper discusses Blackbird's design, principles, and use cases.**

Within a year of its completion, six projects have begun using Blackbird. The Mars 2020 mission is using Blackbird to generate command sequences for cruise and Mars approach. By using multi-mission models, the Mars 2020 cruise adaptation was created in fewer than three months by three engineers at less than half time each. Work has begun to use Blackbird for communications planning during Mars 2020 surface operations. The Psyche mission uses Blackbird to generate its reference mission plans in development. Full simulations with 123,000 activities and 4.7 million resource value changes complete in about one minute. Psyche is also working towards using Blackbird in operations to support integrated activity planning and generate sequences. The InSight project is using Blackbird for mission planning in operations, replacing error-prone manual processes. For the NISAR mission, Blackbird evaluates threats to the commissioning phase timeline. The Europa Lander pre-project used Blackbird to perform a trade study. The ASTERIA mission is automating sequence generation in Blackbird. Going forward, more interested projects are likely to begin using Blackbird, and the capabilities of the core and multi-mission models will keep growing.

TABLE OF CONTENTS

1. INTRODUCTION

Every deep-space mission must solve a variety of planning problems throughout development and operations, and systems-level simulation is typically employed to validate the resulting plans. While the problems are unique to each mission, there are typically broad similarities between them that have driven the creation of persistent and sometimes even multi-mission planning frameworks whose goals are to best enable accurate and timely formulation and disposition of plans. One common trait among these is that they encapsulate spacecraft behaviors as 'activity types', which define to the simulation engine the effect an activity instance of that type should have on the results. Activities can write commands to output sequences to send to the spacecraft, impact resources such as battery state of charge or data volume, or give a user context they can use to make better decisions. The choice of how to define activity types is important, since the definitions should be intuitive and easy to modify, easily understood by the simulation engine, and expressive enough to fully capture the effects of the activity.

Most activity planning software in use at NASA's Jet Propulsion Laboratory (JPL) and elsewhere in the space operations community, such as APGen [1], ASPEN [2], MSLICE [3], COCPIT, SPIKE [4], PAGE, MAPPS [5], and more, use domain-specific languages (DSLs) or attribute-based schemas to define constraints on activities and their behavior. This reliance on interpreting custom languages poses user training, core software maintenance, performance, extensibility, and other challenges, as explained in a following section. There have been

978-1-7821-2734-7/20/$31.00 ©2020 IEEE

frameworks that constructed parts of this definition in standard programming languages – for instance, the MOST scheduler used by the Philae team in 2015 was backed by the ILOG Scheduler, which places activities based on constraints defined in standard C++ [6]. The planning tool CLASP being used for the NASA-ISRO Synthetic Aperture Radar (NISAR) mission's science phase uses a mixture, where spacecraft information is defined in a schema while custom scheduler logic can be written in C++ [7]. In 2017, Pierre Maldague proposed an extension to APGen called A++, where the user would define each Turing-complete activity in C++. In early testing, this change resulted in an order of magnitude speed improvement and demonstrated the potential of defining behavior in a standard programming language.

In part inspired by that work, in early 2018 a team of planning and software engineers at JPL began working on a new simulation environment that was to be as lightweight and easy to maintain as possible with an emphasis on user experience. Out of that effort came Blackbird, a Java-based object-oriented discrete event simulator intended as a multi-mission spacecraft simulation framework that enables engineers to write code in a standard language that is executed directly by an industry-standard environment. Within a year of its introduction, six projects had begun to use Blackbird to solve a variety of critical planning problems. This paper discusses the research and principles behind, the design of, and the mission use cases of Blackbird.

2. User Research

As the idea for Blackbird began to solidify, the development team performed extensive research that took into consideration the use cases of seven JPL missions. This section presents some of the ideas learned from that effort and the resulting actions taken.

Problems with Schema and DSL-based Activity Definitions

When implemented well, DSLs and schemas allow for users to provide specifications for a model with significantly less code, which can reduce cost and make the code more readable [8]. The use of a DSL can also reduce the computer science knowledge needed by adapters. However, the use of DSLs or schemas shifts the responsibility for the optimization, maintenance, and development more onto the developers of the core software, which leads to significantly more challenging and costly development of the core software [9]. Over time, the increased difficulty maintaining the software leads to challenges finding developers to maintain and improve the core, which in turn makes the software more painful to use by mission adapters.

Writing code for planning and simulation frequently requires the use of complex logic, which includes the use of data structures, conditionals, loops, functions, and calls to external models or scripts. However, both schemas and DSLs limit the capabilities offered to adapters, and any increase in capability requires significant effort on the part of the developers of the core software. For example, a schema for defining a data model might start with an attribute for "amount of data generated" for a given activity. But then, a requirement might get added for representing data volume generated given a data rate and a period of time. Now, attributes must be added for data rates and durations. Next, a requirement could get added that an activity should be able to look up the data generated in a table, or call a script to calculate the data generated by an activity. These feature requests are very common in planning and simulation, and each mission using a schema for this purpose needs to account for them. Over time, there is a trend towards arbitrarily complex behavior needing to be represented, which can lead to hard-to-maintain workarounds if the core code is not designed with this idea in mind.

The Mars Science Laboratory (MSL) and Mars 2020 missions both use schemas in their activity dictionaries, which define the model used for planning and simulation. MSL's MSLICE software uses an Extensible Markup Language (XML)-based schema to define activities and models. However, even after adding several capabilities to the MSLICE schema, a feature to inject arbitrary JavaScript code into the XML was required to meet all of the feature requests. This made the schema very difficult to work with, both from a usability and maintainability perspective, since that JavaScript cannot be debugged normally. The Mars 2020 software COCPIT is still in development, but a JavaScript capability has already been added into its schema for math calculations.

The APGen planning software, which has been used successfully for 10 missions in operations and development, has a DSL which evolved over many years due to similar scope increases. It too started as a schema-like representation of activities and their simple impacts to resources, but evolved over time into a Turing complete language capable of arbitrary complexity [1]. Despite adding this capability, mission adapters still have to write a significant amount of code to replicate functions available in libraries found in common languages. For example, APGen does not have uppercase or lowercase methods, so adapters need to define their own functions which parse a string and return an uppercase or lowercase version of it. There are more complicated cases of this that can involve thousands of lines of code. In APGen adaptations, methods for simple conversions are sometimes duplicated under different names due to lack of standard libraries and the complexity of the custom adaptation. As will be discussed later in the paper, the performance of these tools has also been an issue.

The scope creep seen in the current DSLs and schemas at JPL makes maintaining them costly, and finding developers with the right expertise is challenging. There is currently only a single APGen core developer, and MSLICE has required multiple developers since it began to be used in operations to support the core software and schema. In addition, training to become a capable adapter of the MSLICE activity dictionary schema takes about six months and requires previous familiarity with the problem space. The transferability of the knowledge gained from working on both the core software or the mission-specific adaptations has also been a factor in finding developers. Using a schema or DSL is specific to the application, and cannot be easily applied to another role, so many developers hesitate before making a long commitment to create or maintain these types of software.

Lastly, the use of a schema can obfuscate the underlying system model, and users may not understand how activity types they are creating impact the larger system, which could lead to incorrect simulation results. A common solution to this, which Mars 2020 has chosen, is to limit the complexity of the system model, but there are many benefits of having a detailed model. For example, Europa Clipper's high-fidelity APGen model has been instrumental in the design of the spacecraft and the mission [10], and would not be possible with a simplistic schema.

JPL Mission Use Cases

Blackbird was designed with a well-understood set of use cases in mind, and priority was given to missions currently in development or operations. Generally, Blackbird use cases can be broken into three major categories: planning, simulation, and sequence generation. Blackbird is not the first project to address these problems, and its developers heavily leveraged the lessons learned from past software.

The first category, planning, is done during all phases of a JPL mission's life cycle. Planning involves the scheduling of various activities or events over a period of time. Prior to mission operations, this work is largely mission planning, which encompasses the higher-level design and scheduling of mission activities. Mission planning has been done by hand for many missions, but newer missions are making use of software to mostly automate this type of work. Mission planning work often continues throughout operations, but in parallel to activity planning, which is done on a shorter time scale with more detail. Activity planning usually involves more human-in-the-loop coordination, but is gradually moving towards letting software optimize a plan, where users provide constraints rather than specific times when activities will occur. Modern operations efforts typically now involve plans that are a combination of automatically placed and user-defined activities. Activity planning software must enable such a mixture to fully support mission design and operations.

The second category, simulation, generally involves representing the behavior of a system over time. Like planning, simulation can be done at many levels of fidelity. The purpose of the simulation is to validate that the plan meets the requirements put on it, and that it will keep the spacecraft in a safe and expected state throughout execution. Thus, the level of detail of the results needed by the project drives the model fidelity, which can evolve along with the mission phase. Simulation is often done in parallel with planning, but does not have to be. The calculations involved in a simulation can often be time-consuming to perform, at times taking several minutes or hours to complete. Common bottlenecks during a simulation include calculating geometric information, thermal modeling, or tracing the execution of commands.

The last category of use cases for Blackbird, sequencing, refers to the auto-generation of sequences of spacecraft commands based on an activity plan. Sequence generation is usually reserved for missions in late development or operations, but it is a mission-critical part of those phases of JPL missions, and must be addressed to successfully operate a spacecraft. Although different JPL spacecraft often have analogous functions, the format and contents of the commands and sequences can be drastically different, due to their connection to spacecraft flight software. The generation of these sequences ranges from completely automated to hand-generated, and the software supporting operations must account for the fact that not all sequences will be generated in the same way.

Planning, simulation and sequencing make up the majority of the uplink portion of spacecraft operations. The other half of mission operations, downlink, involves analyzing telemetry and assessing spacecraft health and safety. There typically is a need to update simulations and activity plans based on spacecraft telemetry. For example, a power model must be updated with a new battery state of charge when telemetry is received from the spacecraft. This handoff between downlink and uplink is a common problem across missions, and must be supported. Uplink software typically does not read spacecraft telemetry directly, but uses curated subsets of it as initial conditions for future simulation runs.

There are two general types of users of planning systems: operators and adapters. Operators are typically systems engineers or engineers with domain-specific knowledge, and will often have little software experience. A common requirement is for these users to be able to interact with a planning system without looking at or writing code. To support the use case of operators, missions always wrap the underlying scheduling and simulation component in a full planning system with graphical user interfaces (GUIs), command line capability, and reporting tools. In order to limit scope, Blackbird was designed with this paradigm in mind, and operators only interface with the larger mission-specific planning system which incorporates Blackbird.

Adapters are users who write mission-specific code which defines the behavior of the spacecraft in a simulation, as well as the activities that can be performed by the spacecraft. Adapters are therefore the type of users who interface directly with Blackbird. Adapters can range from having a strong background in computer science to having no prior exposure to software engineering, as is the case for many engineers defining models in Blackbird. There are some cases where adapters also go on to be some of the operators of the planning system.

Mission-Driven Development of Blackbird

The core code of Blackbird was designed and completed over a period of four months, with another four months creating multi-mission spacecraft models and proof-of-concept mission-specific adaptations. Within one year of that, six missions had already begun to use Blackbird to solve a variety of critical problems. This rapid development was made possible by an agile development process and developing entirely from the bottom up. Throughout its development, parts of the code were being refined as requirements evolved, and the code was developed with the expectation that future changes would be required.

The scope of development and the deadlines were driven by JPL mission needs, with fixed schedules from launch dates and critical reviews, leaving no room for delays in the deliveries. The main requirements of Blackbird were driven by the use cases of the Psyche, Europa Lander, Europa Clipper, Interior Exploration using Seismic Investigations Geodesy and Heat Transport (InSight), Mars 2020, Juno, and Mars Reconnaissance Orbiter (MRO) missions, and all of the features necessary for these missions were completed in the first four months of development. During and after this process, it was required to demonstrate that Blackbird was an improvement over previous software in accuracy, capability, ease of use, performance, and maintainability before it could be adopted by missions.

Early on in the development of Blackbird, extensive interviews with operators and developers on the missions listed above took place. In addition to this, the developers of Blackbird had worked on the APGen adaptations of several of these missions, which contributed to the overall requirements and concept of operations. Across all operators and developers of JPL planning and simulation software, the most common theme was the importance of usability, which was mentioned even more than performance or maintainability. In addition to this, operators expressed frustration when the software was designed with specific assumptions about the way it would be used, which often led to arduous workarounds to make the software meet common use cases on missions. Blackbird was designed to avoid these sorts of assumptions, and leaves room for mission adapters to extend the framework to meet any need.

3. BLACKBIRD SOFTWARE DESIGN

Design Principles

During its development and completion, the Blackbird developers had in mind some principles which ended up being the key to its rapid success.

a. Programmer efficiency is not only a nice-to-have—Any other criteria one could use for evaluating a framework like security, performance, extensibility, etc., are so enhanced by developers being able to work efficiently that it is of paramount importance to create an intuitive development environment. All the mission work mentioned in the rest of the paper was only possible to do quickly because Blackbird adheres to this principle.

b. Gall's Law—"A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system" [11]. Blackbird was designed to be as simple as possible; the core codebase is less than 10,000 lines of code including frequent comments and whitespace, whereas other planning tools like APGen, ASPEN, and MSLICE all have well over 100,000 lines of code in their cores. The simplicity meant the program was quick to form and start meeting mission needs. This simplicity also translates directly into maintainability, since a new developer will have much less to learn. A common reason for the gradual degradation of multi-mission tools is having 5-12 developers build the system then only keeping one developer once it is in 'maintenance mode'. Blackbird was primarily built by two developers in four months at half-time each, so there will be less of this kind of drop-off issue.

c. Design from the bottom-up, not top-down—Mission planners and schedulers are the ones who know their needs best, and so the top priority should be meeting those and not trying to impose a technology then fitting it to the need. Blackbird was developed grassroots from the people doing development and operations work.

d. Inherit Proven Ideas, Not Legacy Code—Blackbird's high-level design and how it conceptualizes activities and resources are inherited from APGen, which has been used successfully on over 10 missions. This similarity lessened the possibility that any key use case was overlooked by the requirements, or that simple ideas were getting over-complicated. However, a conscious decision was made not to double or triple the size of the code base in order to be backwards compatible and support adaptations written in the APGen DSL. In addition to making the core significantly less maintainable, supporting the DSL would hinder the transition to an efficient development. The need to rewrite key multi-mission models was a positive impetus to improve their code quality and make them more performant.

e. Make Truly Multi-Mission Code—Write every piece of potentially multi-mission code as if you personally will have to use it to support multiple missions. This means writing requirements from the holistic combination of all potential customer missions' use cases, not taking literal requirements from any one mission. Design interfaces so that mission engineers can do planning how they feel most comfortable, instead of presupposing how a mission should do planning. Sections 4 and 5 will detail the time saved by multiple missions using the same code. Blackbird itself owes its viability to open source software, as well as the multi-mission tool Resource and Activity Visualization Engine (RAVEN), which was built before Blackbird existed but was generic enough to be extremely useful to Blackbird users without any modifications.

f. Avoid Boilerplate and Favor Readable Code—Help developers writing in your framework create as little non-business-logic as possible.

Blackbird Overview

Blackbird 'core' is a discrete event simulation engine with scheduling and sequence generation capabilities. The events that are added to the timeline are Java thread objects, which when executed contain activity behavior defined in specific ways. The Blackbird framework provides basic classes that engineers can extend; those base classes contain minimalistic machinery that governs the order in which activity instances are processed, how information appears in output files, how one can interact with the model, and more. One partially analogous framework to Blackbird is MATLAB, which provides useful extra functions for common tasks in a specific domain, while also providing an execution environment that processes the defined constructs.

An 'adaptation' refers to a set of subclasses that define the behaviors of a spacecraft and their effects. In addition to those classes, missions have choices for how they interface with the .jar compiled file. A typical output format is a standard Time-Ordered-List (TOL), which can contain all the activities and resource changes in the engine's memory.

Framework Constructs

The two classes most fundamental to Blackbird are Activity and Resource. Most of the work of an adapter is extending Activity to create specialized activities that set Resource objects appropriately. A plan consists of a set of time-tagged Activity instances, and the simulation results are the Resource value histories throughout the plan.

An Activity is a human-level abstraction of a spacecraft behavior, or a behavior that affects the spacecraft. Spacecraft behaviors are processes like performing a Trajectory Correction Maneuver, beginning transmitting data to earth, or collecting science data. The environment can influence the spacecraft in ways such as ground station availability, the amount of charged particle impingement on solar arrays, or temperatures that affect how much heater energy the spacecraft will need. For clarity, one can speak of 'activity types' and 'activity instances.' An activity type is the definition of what the planning engine should do when an activity instance of that type is placed. An activity instance is the union of an activity type, a start time, and any other parameters the type requires to know how to behave. In order to create an activity type in Blackbird, one must extend an Activity superclass and implement between one and five methods: model() controls how activities impact resources, decompose() provides a way to create child activities outside of simulated time, dispatchOnCondition() provides a way to create child activities during simulation, setCondition() controls the condition for the dispatch, and sequence() controls how the activity writes to sequences. The methods available in the Activity class equally support multiple scheduling paradigms: one where adapters explicitly specify the algorithm by which to place child activities, another where adapters tie activity creation to resource changes, and a third where requests and constraints are passed to a depth-first-search based constraint satisfier.

Resources should be thought of as variables that remember the times they were changed throughout the simulation. The class encapsulates the value, provides the equivalent of getter and setter methods, and provides the automation to write out its time history to the core output mechanisms. Due to the generic setup, any data type can be tracked in a resource, from Booleans to Times to quaternions. Blackbird also supports arrayed resources and automated resources that track values and update without needing an Activity.

Blackbird supports active and passive constraint checking. Active constraint-based scheduling is done using the Condition class, which represents a resource comparison to a threshold. The Constraint class does passive constraint checking and is more useful for manually created plans that are not valid by construction. The Window class represents a beginning and end Time, and encapsulating it makes many common planning paradigms much more natural. To produce sequences, the Sequence class is also important.

Notation Introduction and Worked Example

Figure 1 contains the symbol glossary that will be used to build the structure diagrams of Blackbird code throughout the rest of the paper. Not all diagrams will be at the same level of detail, but ideally having a common icon set improves comprehension. These symbols may be color-coded to indicate multi-mission components or those shared by multiple adaptations within a single mission.

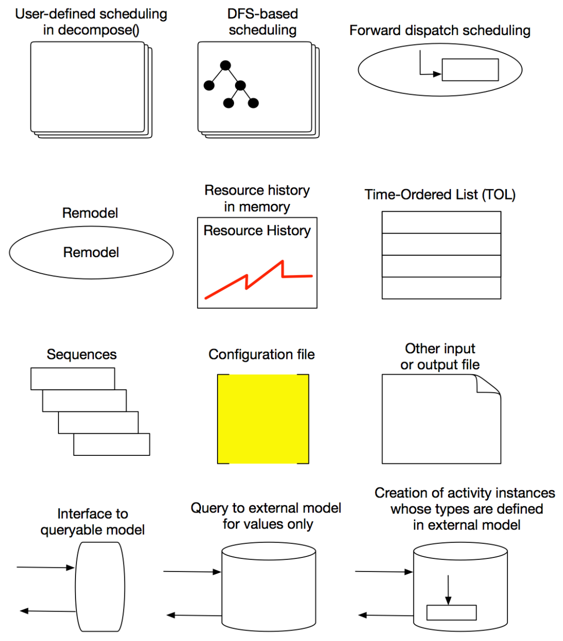

User-defined scheduling in decompose()
DFS-based scheduling
Forward dispatch scheduling
Remodel
Remodel
Resource history in memory
Resource History
Time-Ordered List (TOL)
Sequences
Configuration file
Other input or output file
Interface to queryable model
Query to external model for values only
Creation of activity instances whose types are defined in external model

Figure 1. Adaptation Symbol Glossary used in diagrams throughout this paper. Each symbol has a short explanation of what it represents above it. Activities that just set resources will be represented as simple rectangles.

Figure 2 represents an example Blackbird run that exercises decomposition, modeling, forward-dispatch scheduling, and sequence generation. The caption interprets the result of sending each command to the engine.

Modularity

Blackbird was designed so that each core class is well encapsulated, so that changes to the internal workings of each class should not affect the behavior of other classes. Each class has a small number of public methods that all other classes must go through, so implementation changes simply have to continue to meet the contract of those public methods. After Blackbird had been completed for a year, there was a desire to replace Blackbird's original Time class with the Java version of the jpl_time library, which used a different backing data structure to store time values. Changing over to the new Time class only required changing 10 non-import lines of code in surrounding classes. As another example, currently Blackbird's ActivityInstanceList is backed by an ArrayList. However, the ActivityInstanceList public methods for iteration, addition, and searching do not depend on that implementation. If in the future the team wanted to change out the backing structure to, for example, a TreeMap for faster searching, or a combination of ArrayList and HashMap for fast iteration and lookup, the code changes would be entirely confined to that less than 200 line-long class and would not affect any other classes in the code.

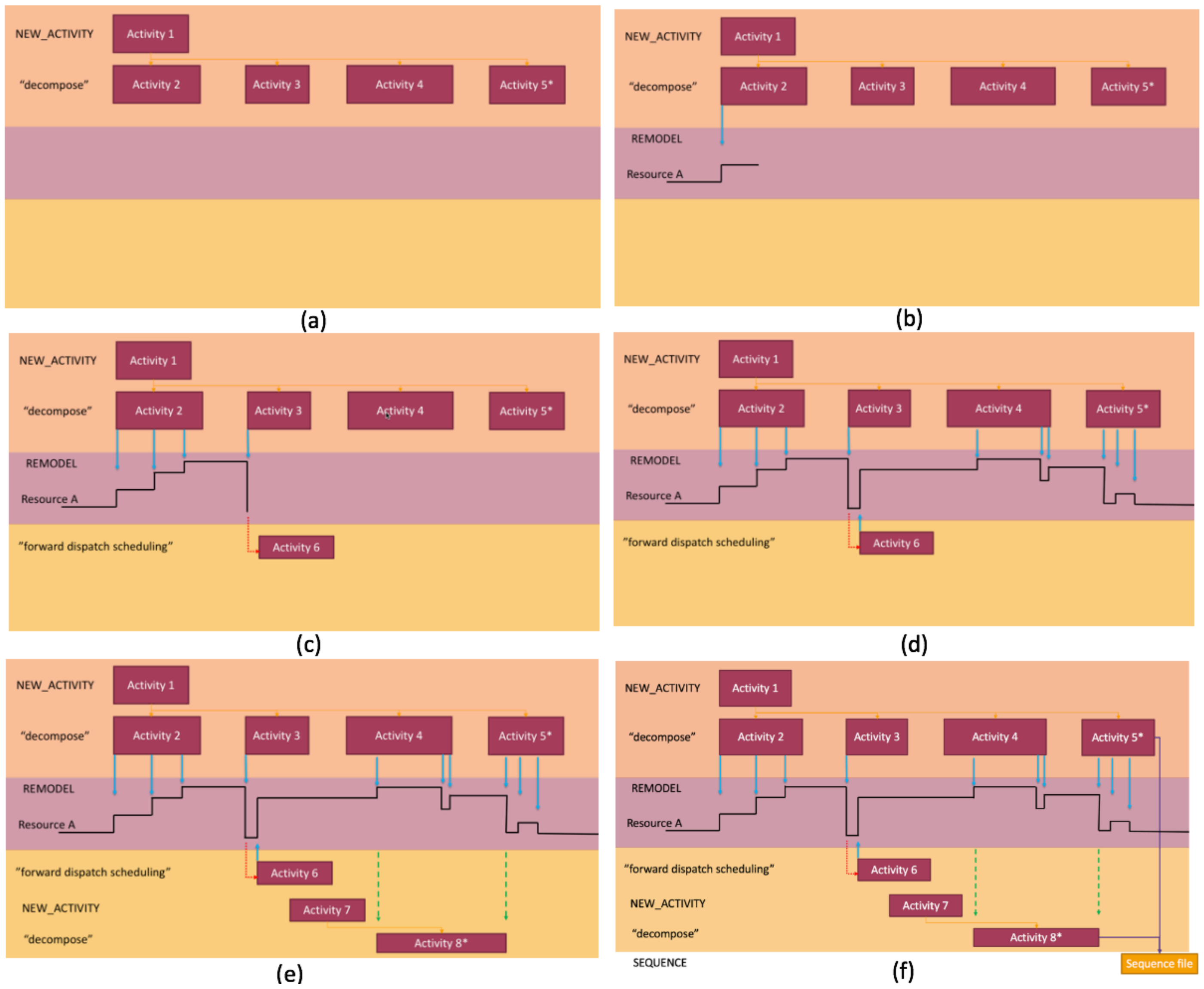


Figure 2. An illustration of the state of the engine after different commands are sent to it.
(a) Activity 1 is placed manually and decomposes into children activities 2-5.
(b) Activities 1-5 are modeled and affect resource A over time.
(c) A forward-dispatch scheduler places activity 6 during modeling when resource A is equal to a specific value.
(d) Activity 6 is now part of the model and immediately starts affecting resource A.
(e) Modeling is over, but resource values persist. Activity 7 is placed manually and decomposes into activity 8.
(f) The sequence command is run. Activities 5 and 8 expand into commands which are a part of a sequence.

Maintainability and Code Health

Blackbird core was developed with current software development best practices in mind. Unit tests provide 80% line coverage of the entire core codebase as of October 2019. All Blackbird changes go through a pull request process where another developer must approve the changes. There are generic regression tests that need to pass before pull requests can be merged, and most missions have mission-specific regression tests that are used to evaluate changes. The static code analyzer SonarQube is regularly run on the code and as of October 2019 it reports a technical debt ratio of less than 5% and zero bugs (there are 8 intended false positives). There are Javadoc-style and inline comments frequently throughout the code. Doxygen, a standard documentation generator, has been set up to create reference pages and dependency diagrams. Mission adapters are encouraged but not forced to follow these best practices.

Blackbird uses standard imports whenever possible and does not use custom data structures. The libraries that Blackbird depends on are all actively maintained and will be for the foreseeable future. The build process is defined in the software management tool Apache Maven and takes about 2 seconds, so rapid iteration is possible. To date, all core changes that missions have requested were implemented between half a day to two weeks after.

Ease of Use

Much of Blackbird's ease of use derives directly from how adapters write code directly in an industry-standard programming language. Features such as an integrated development environment (IDE) with debugging, static code analysis, performance profiling, ease of calling external libraries, online reference resources, native test frameworks, formal dependency management, and more have all reduced development time and uncaught errors.

Java has been GitHub's second most-used language since 2013 [13] and the first most-used on the TIOBE index since 2015 [14], so new developers may already be familiar with it, and all adapters benefit from an active community and helpful resources. To create an activity in Blackbird, only a small subset of the full Java language is required, so even new adapters can make contributions quickly, then add more complex behavior as their proficiency grows. Blackbird adapters typically become proficient in about two weeks full-time. Blackbird adaptations listed in later sections all required people working only part-time for limited commitments, whereas traditionally missions need to dedicate at least one full-time person for years. If a project decides that they want to use another language to write their activities, they can seamlessly incorporate Python-like Groovy, JavaScript-like Kotlin, or Lisp-like Clojure, which can all inherit Blackbird's Java base classes. Groovy and Kotlin proofs of concept were built which motivated multiple projects using Blackbird to consider partially switching to one of these other languages to further improve the new adapter experience.

Based on experience training more than 10 people to use Blackbird, the largest impediments to ease of use are not the language syntax, but the use of an IDE for adapters without prior experience, and understanding when their adaptation code will execute as part of the simulation. One is not forced to use an IDE to develop in Blackbird, but the initial investment to learn to use one is often worth the reduced risk and improved productivity afterwards. Stepping through code in the debugger has greatly helped adapters understand when their code will be called, which will always be non-trivial for any complex system. There is a detailed adapter's guide which walks new users through defining and deploying an adaptation.

Performance

Blackbird v2018-09 was put through a number of benchmarking tests to ensure that its performance would meet mission needs. It was primarily compared with APGen v34.5, since that is the tool the customer missions would have used if Blackbird were not available. The most intensive test involved both engines running exactly the same adaptation such that the time-ordered list output files, which contain the complete state of the engine at the end of execution, produced by APGen and Blackbird were exactly the same. This test exercised decomposition, modeling, function calls, referencing past resource values, constraint checking, pausing activity instances and resuming them later in simulated time, referencing global/public static variables, arithmetic, parameter passing, and file writing. The number of activities and resource usages were varied and the speed and memory impacts of that change was recorded. The results of the test are shown in Figure 3.

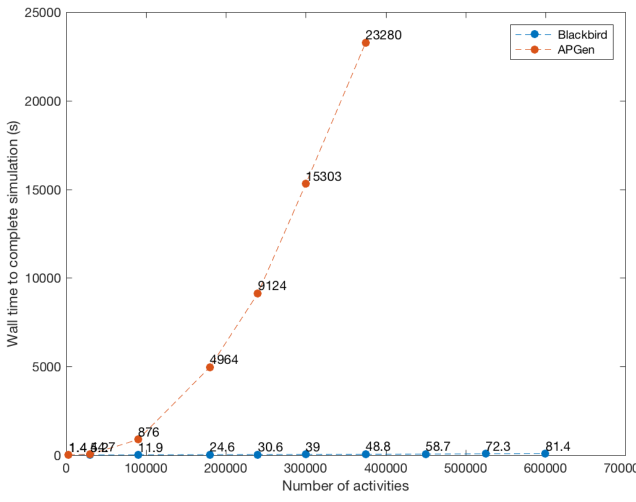


Figure 3. Speed scaling of APGen and Blackbird for representative identical test activities.

In terms of speed, Blackbird scales with O(n) where n is the number of activities and resource usage nodes, whereas APGen scales with at least O(n^2). For reference, large adaptations typically produce 150,000 activities that are much more complex than these test ones, creating millions of resource records. At about that number of activities, APGen took about an hour to finish the simulation and write the file, whereas Blackbird took about 20 seconds. In terms of memory consumption, Blackbird scales with O(log(n)) in terms of number of activities and resource usage nodes, whereas APGen scales with O(n).

Missions are seeing this speed increase for their real applications. When comparing the production APGen and test Blackbird Insight Cruise adaptations, it took Blackbird 11 seconds to produce identical sequences as APGen, which took 49 seconds on the same machine, which is about 5x faster than the machine actually used in operations. The memory used by Blackbird was 155MB while 282MB was used by APGen. The speed it takes each mission's adaptation to run is listed in the following sections, but none of them take over one minute, except for geometry resource generation for Psyche, which takes 8 minutes. A similarly complex geometry computation for a Europa Clipper task takes APGen multiple times longer.

At least four factors contribute to these large differences in run speed and memory usage.

a. Lack of Custom Interpreter Overhead—Blackbird allows simulation code to be executed directly by the Java Virtual Machine, which has been optimized for two decades by teams of programming language and compiler experts. A naïve interpreter executes a DSL's instructions without branch prediction, loop unrolling, function inlining, and other industry standard compilation techniques [15].

b. Data Structure Scaling—Blackbird's internal data structures were carefully chosen for optimal speed and memory footprint. As an important example, its resource histories are stored in a Java-standard red-black tree (TreeMap), which provides the O(log(n)) search for ordered data while providing O(log(n)) insertion. In contrast, APGen's resource histories are stored in a custom data type that degenerates to O(n) iteration for common operations.

c. Object-Oriented Referencing—In Blackbird, all internal structures and adaptation data are objects, so they can directly invoke methods of each other, whereas with a DSL events must be passed back to a monolithic interpreter which then decides what to invoke.

d. Optimized Adapter Algorithms—Blackbird offers adapters freedom to optimize their code. While there are limited data structures available to adapters within a DSL, Blackbird adapters can easily use efficient data structures. In the APGen InSight cruise adaptation, there are several resources which have their values set by listening for changes to other resources. Each time one of the resources changes value, the algorithm in APGen loops through all resources being watched and checks their state. In Blackbird, custom automated resources keep track of the number of other resources which are in a certain state, removing the loop. This leads to optimized resources which are up to 50 times faster to evaluate in Blackbird than in APGen.

4. Multi-Mission Models and Patterns

Adapters working in the Blackbird framework have prioritized creating multi-mission code, which have grown into sophisticated and well-documented models that make it easy for missions to start doing complex work immediately. This section also presents design patterns that have emerged naturally in multiple adaptations based on the tools the framework provides.

Multi-Mission Models

The Blackbird core software itself knows almost nothing about spacecraft – it could be used as part of a planning system in many different industries. When a new mission adopts Blackbird, it is desirable to have to reinvent as little of the wheel as possible and learn from past missions. Knowledge about spacecraft and infrastructure behavior is thus encoded in a set of multi-mission models, each of which is responsible for a different typical subsystem. For multiple adaptations, the multi-mission models are capable enough that the mission-specific code required is much smaller than the amount of multi-mission code the adaptation depends on. The list of multi-mission subsystems follows the set of APGen models that most notably Europa Clipper is currently using [10]: ground station, geometry, data, power, telecommunications, guidance navigation and control (GNC), propulsion, and more.

In addition to the models, by which is meant the implementations of resource-setting and scheduling code for these subsystems, the Blackbird multi-mission project aims to provide a multi-mission Java interface for each subsystem that defines the methods needed to interact with a model of it. This way, even if no multi-mission model is provided for that subsystem currently, mission adaptations are forward-compatible with moves to higher-fidelity models with no change to code for any activity types. This kind of encapsulation is only possible because the framework is object-oriented and flexible.

The Blackbird multi-mission ground station model is the most sophisticated model in the current suite, and is the most critical for sequence generation tasks, which is one of the fundamental Blackbird use cases. The model is composed of 6,000 lines of code, which is about 2/3rd the size of Blackbird core itself. The ground station model offers the following capabilities for missions: view period and elevation generation from a variety of inputs, station allocation parsing and comprehension, spacecraft doppler mode transition tracking, spacecraft antenna state model, station allocation generation based on a depth-first-search constraint solver, output to common DSN formats, and built-in information about DSN stations. The Blackbird implementation of these is partially based on the corresponding APGen model, but was completely restructured for maintainability and is more advanced; for example, the Blackbird ground station model processes allocations for Delta-Differential One-way Ranging (delta-DORs) correctly without the need for editing input files, and the APGen model does not allow for multi-mission parameterized track generation. Figure 4 diagrams the structure of the DSN multi-mission model when reading in files and providing doppler mode information to mission-specific schedulers.

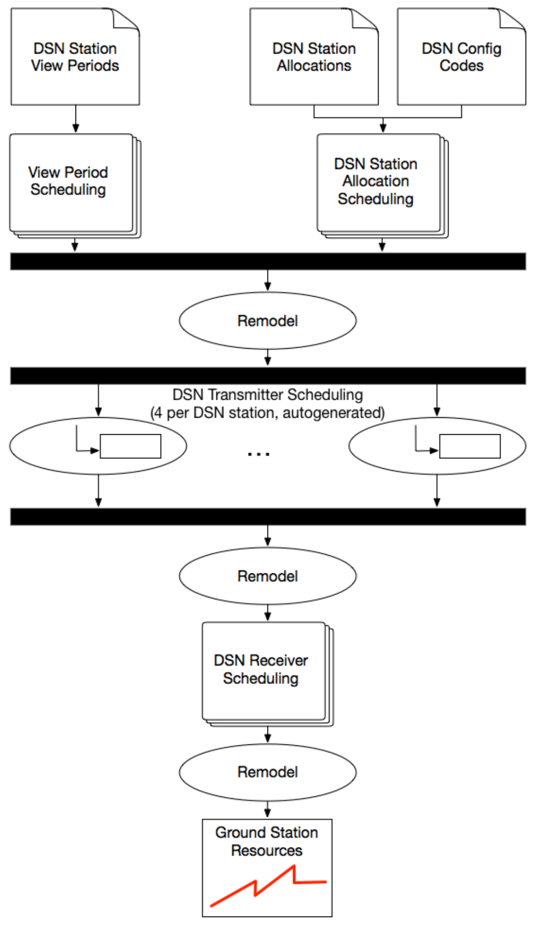


Figure 4. Structure of section of multi-mission ground station model that simulates doppler transitions

The Blackbird multi-mission geometry model is another critical model, particularly to provide context and schedule activities in mission planning work. It makes heavy use of Blackbird's native integration with NASA's Navigation and Ancillary Information Facility (NAIF)'s Spacecraft ephemeris, Planetary ephemeris, Instruments, C Pointing Matrix, Event Info (SPICE) library in order to calculate geometric quantities of interest, such as: upleg and downleg time, spacecraft position and velocity with respect to different bodies, spacecraft orbit parameters, angles between different bodies or points on bodies, coordinates of subpoints on bodies, eclipse windows, occultation windows, and more. Each set of quantities can be queried at user-specified timesteps, which can be fixed or variable so more samples are generated around less linear regions. For the latter, a desired tolerance can be provided that feeds into the determination of when the next timestep should be. Typically for mission planning work, once all geometry is computed, interpolating between points is sufficient, but those building operational tools would likely favor querying SPICE at exactly the time they are interested in.

The Blackbird data model provided is medium-fidelity and conceptually similar to data models most APGen adaptations employ. Data is generated at set rates inside various prioritized bins, and activities can change those rates or make sudden changes to the volumes in the bins. For most missions to date, that level of detail is all that could be tracked due to imprecision in activity definitions and generation of simulation initial condition files. The main purpose of the model is not to track all data products onboard at any given time, but for each downlink pass provide an estimate of how much data is expected to be downlinked from what instruments, and how much will be left onboard.

The multi-mission models live in a different repository from Blackbird core and are packaged separately, so missions can import only the interfaces and models that will be useful to them. All the interfaces in the multi-mission model repository have complete Javadoc comments, and Doxygen has also been run over the multi-mission models. The resulting diagrams are progress towards the goal of new adapters of making the multi-mission models as standard and easy to use as possible. As developers on missions write code that could be multi-mission, they are encouraged to contribute back to the multi-mission repository, and regression tests are used to make sure updates do not break use cases of the missions already using the model in question. All of the Blackbird core and multi-mission model code is available to everyone at JPL, and so are the largest mission-specific adaptations. This openness facilitates the sharing of models.

Design Patterns

Through working on APGEN adaptations, Blackbird's multi-mission models and the mission work listed in the next section, there have emerged common design patterns that would need to be supported by any similar software.

Dependency Layers—One common pattern is dividing the calculations up into dependency layers. Systems engineers are already used to thinking about what information needs to be present for the next step in a process to execute, so this has come naturally to adapters thus far. Each layer comprises a set of decompose() methods that create activities whose placement does not depend on each other, and then a remodel which runs the placed activities and sets resources so that the next set of resource-aware decompose() methods can run. Each layer of decomposition runs only once exactly when specified, which is performant and intuitive. Furthermore, this pattern provides an easy method to start from a known point later, since intermediate layers of activities can be saved off to a file then read back in before resuming scheduling.

Generator-Type Activities—Blackbird is a discrete event simulator, but many models or external programs it must interact with use fixed time steps. Additionally, even if information is not being used regularly to calculate other products, one might wish to write out values every so often for display and review purposes. A simple pattern to bridge the two is to have a 'Generator' activity with a model() section that contains a loop, where each loop iteration waits until a later simulated time. The amount the method waits can be fixed for a fixed timestep calculation, or variable. At each simulated time the loop stops at, the method can query or direct a model, and/or calculate a resource value and store it.

Visualization-Only Activities—Particularly in mission planning but even in operations, there are situations where extra code has to be run to create activities that will simply provide context for the user, not affect any other activities or resources. In some cases, these can make up a significant fraction of the total number of activities in the timeline.

Generic Activities—Missions experience diminishing returns trying to capture the behaviors that may only be performed once, or have negligible impact on resources. In these situations, a useful pattern is to create a 'Generic' activity type, where adapters set up parameterized high-level behavior that fulfills all the goals of the adaptation. These generic activities let planners fill in those parameters for each instance as they would for a schema-defined activity. Common parameters for generic activities include amount of energy used, data volume used, unique name, and what sequence executes them, but specifics vary by mission.

Configuration of Schedulers—The key to not having to make planning code changes when requirements change in operations is to have well-parameterized activities and schedulers. Parameterization also makes it easier to perform trade studies in development. Parameterization of schedulers is typically handled differently than for simple resource-using activities. For simple activities, the constructor accepts parameter values that could be edited in a GUI via a typical edit activity panel. However, schedulers' placement of child activities can depend on a large number of variables, some of which multiple schedulers are referencing, which makes constructors extremely brittle and constricts the flow of information around the simulation. Furthermore, these variables may be complicated data structures that are more easily typed in a different format than the most convenient structure to do calculations. A common pattern to address these issues is to give static parameters values in a text input configuration file, which must be read in prior to executing the schedulers. This reading in process can also be used to convert information from formats easiest for humans to input into more efficient data structures. Blackbird also has a native SET_PARAMETER command that can change public static variables in the adaptation so that what-if scenarios can be run when Blackbird is connected to an external GUI without having to go back out to files.

5. BLACKBIRD MISSION SUCCESSES

Blackbird has been adopted by six missions that span from cubesats to flagships, and it has consistently decreased cost and risk on each of them to date, in some cases significantly. The framework has proved so natural and flexible that it is not just being used in the traditional centralized activity planning and sequence generation roles, but also for tasks that until now have been done manually in Excel or by homegrown scripts where it would not have been worth working in a harder-to-use formal activity planning system.

Mars 2020

Mars 2020 is an upcoming Mars rover mission which will launch in July 2020. The mission has an 8-month cruise to Mars, before landing in February 2021. Mars 2020 became one of the first missions to use Blackbird, and is doing so for both cruise and surface operations, for three functions: generating command sequences for the cruise and Mars approach, generating Deep Space Network (DSN) related input files for cruise and surface development, and automating mission planning during surface operations. Blackbird has already shown successful results for cruise thread tests, which are medium-scale tests designed to exercise a specific aspect of the development, and received an award for its use on the mission.

During Mars 2020's cruise to Mars, commands are sent from Earth which are necessary to control the spacecraft. There are several types of commands sent, but the most common types are for communicating with Earth and performing routine diagnostic and checkout activities. These commands were originally planned to be generated in a largely manual way for Mars 2020, but the project decided to fully automate their generation using Blackbird. This investment resulted in a decrease in the amount of time taken to produce a single sequence from one or two weeks to less than five seconds. This automation not only reduced the staffing required to operate during cruise, but also reduces the risk of human error, due to the numerous calculations needed to correctly determine the timing of the spacecraft commands to communicate with Earth. The Mars 2020 cruise adaptation was developed in less than three months by three part-time recent college graduates, and replaced a schema-based software which provided less automation that had been in development for more than six months. During an early Mars 2020 cruise thread test, Blackbird was identified as the easiest 2020 tool to use by operators with little coding experience.

During the mission phase "Approach, Entry, Descent and Landing" (AEDL), the spacecraft requires different commands to communicate with Earth than used in the cruise phase, and the logic to generate them is also different. However, a large portion of the code can be shared between the two adaptations. There are also multi-mission models used by the Mars 2020 adaptations which were validated during parallel operations with InSight during its cruise.

The cruise adaptation reads in several input files representing geometry and times when the DSN is allocated to communicate with Mars 2020. The adaptation then calls the multi-mission ground station model to calculate resources necessary to show when spacecraft doppler modes change and which DSN stations are in view at a given time. Afterwards, a series of adapter-defined schedulers are run which provide information such as changing data rates or periods when a delta-DOR event will take place. Next, a scheduler to calculate the periods when communication will occur is run, which generates the final activity plan. Finally, the sequences and resources are written out to several files, before they are then validated and radiated to the spacecraft. Figure 5 shows an output activity plan from the code, and Figure 6 shows the structure of the Cruise adaptation.

The Mars 2020 AEDL adaptation is similar, but requires different scheduling logic for the commands which are generated. As shown below in Figure 7, very little code needed to be written to produce different commands and communication behavior, since the multi-mission ground station model and the majority of the cruise adaptation schedulers can be shared across mission adaptations.

In addition to the Mars 2020 cruise and AEDL adaptations, Blackbird is currently being used on the mission to generate geometry and DSN-related files for thread tests. These files are called the DSN View Periods file (VP) and the Station Allocation File (SAF), and they represent when each DSN station is in view of the spacecraft and when the spacecraft is allocated for communication with a given station. These files can take more than five hours for a trained DSN scheduler to create, which quickly adds up for repeated tests. The Blackbird adaptation is only a few hundred lines of code due to use of the multi-mission DSN track scheduler, but can generate new test cases for any time period in seconds. This small adaptation has been used most recently for the launch thread test, but will soon be used for cruise and AEDL tests. There is also a need for this capability during the surface phase for testing. This adaptation is still in development, but is planned for use in early 2020.

Another planned use of Blackbird for Mars 2020 is for the mission planning communication process during operations. There is a need during surface operations to schedule not only when the spacecraft communicates directly with Earth, but also when it communicates with the current fleet of Mars orbiters. Work on this adaptation is beginning, and Blackbird should be able to support a more robust and easier to implement solution than custom scripting.

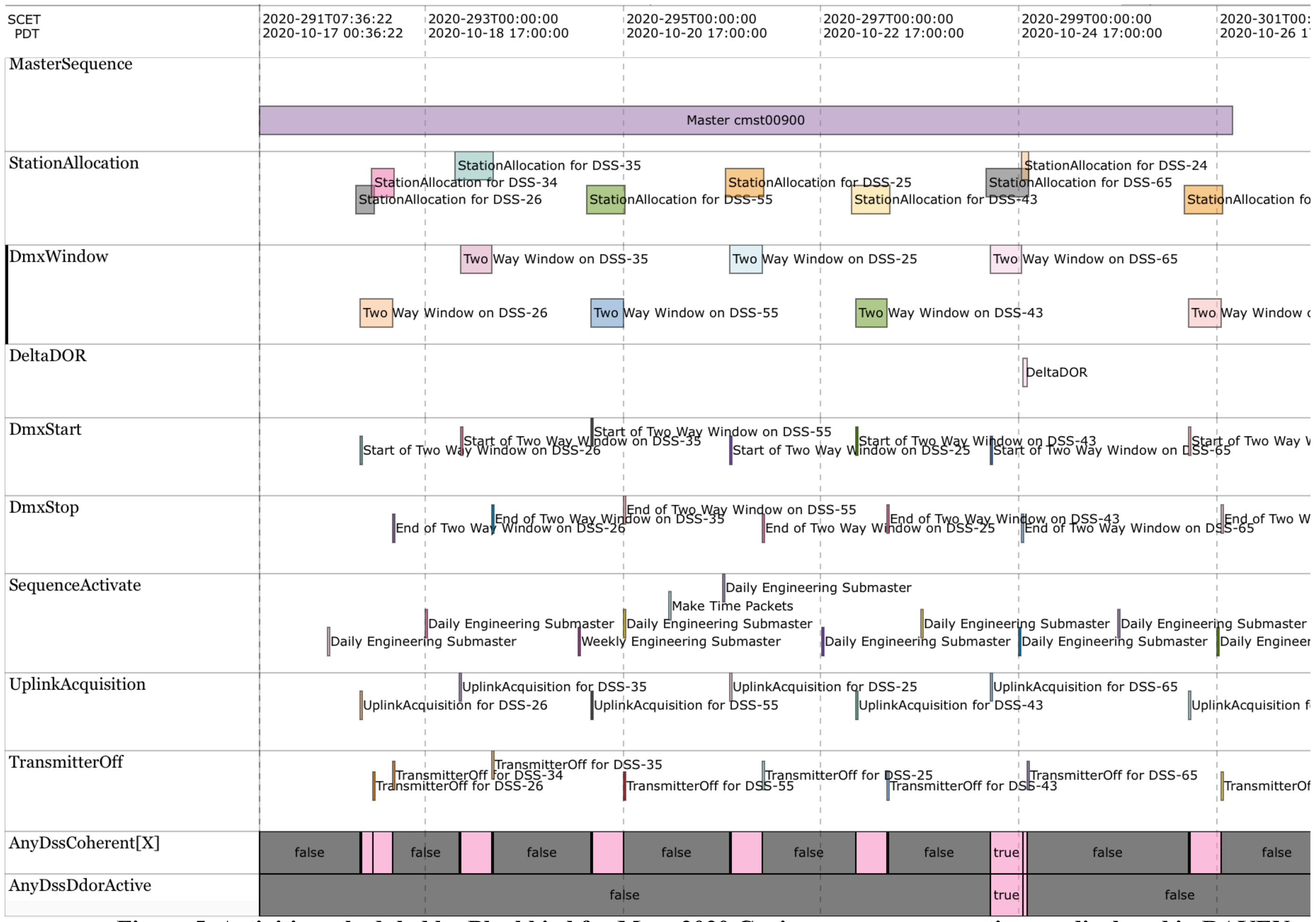


Figure 5. Activities scheduled by Blackbird for Mars 2020 Cruise sequence generation, as displayed in RAVEN

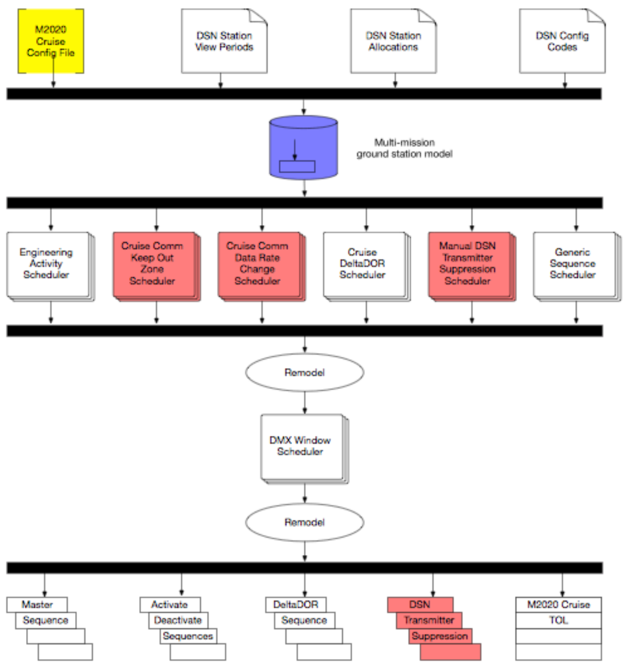


Figure 6. Structure of Mars 2020 Cruise adaptation. Red components are shared between the Mars 2020 adaptations, and blue components represent multi-mission

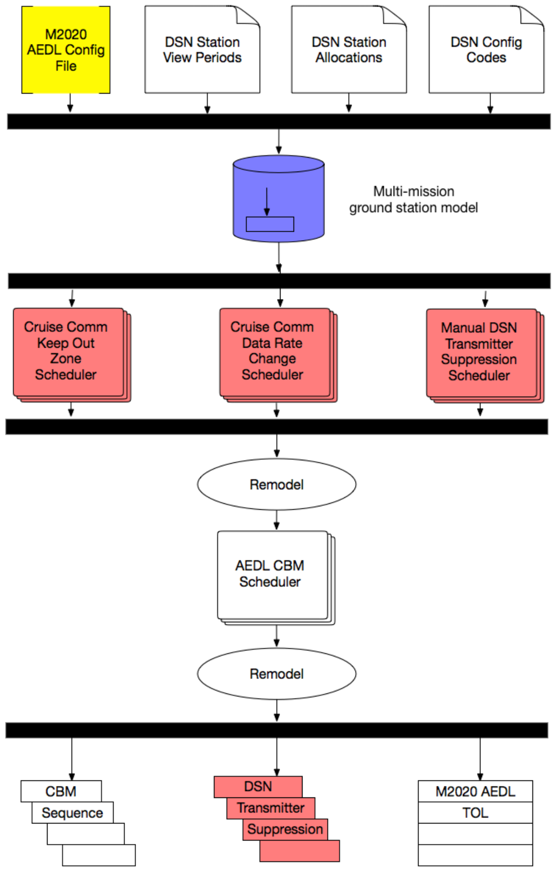


Figure 7. Structure of Mars 2020 AEDL adaptation. Red components are shared between the Mars 2020 adaptations, and blue components represent multi-mission models.

Psyche

Psyche is a Discovery-class mission that will launch a solar electric propulsion spacecraft in 2022 to the asteroid (16) Psyche, a particularly metallic small body where much can be learned about the history of the early solar system. After a three and a half-year cruise phase including a Mars Gravity Assist, the spacecraft will arrive at the asteroid in 2026, bringing with it an imager, a magnetometer, a gamma ray and neutron spectrometer, and a high-gain antenna (HGA) with which to perform gravity science measurements. The operations concept is relatively simple and is based on Dawn's, which had a highly successful low-thrust mission to two planet-like bodies in the asteroid belt [16]. However, the Psyche team desires to improve on the Dawn process where prudent in order to decrease risk and cost in operations. To respond to issues caused by a lack of a formal mission planning function during Dawn operations, Psyche's mission system anticipates having a mission planner continue into operations, and to use an integrated activity planning tool to reduce manual effort and manage potential conflicts earlier. As a Discovery-class mission, Psyche does not have the resources to build an ambitious planning tool by itself, like flagships frequently do. The mission chose to use Blackbird to do mission planning scheduling and simulation in development, and is currently defining Blackbird's role in operations and what interfaces it will need to have with other tools.

Development—In Phases B and C, Psyche's mission planning team has built an adaptation that, starting from just SPICE kernels and a small list of manually planned activities, produces the reference Mission Plan Timeline, as displayed in Figure 8. This timeline includes activity placement down to the level of individual images or DSN tracks, resource histories such as the projected onboard data volume throughout the entire mission, and produces reports that check if plan constraints are satisfied. Each run of the simulation produces 123,000 activities, like science periods, Deep Space Optical Communications (DSOC) opportunities, reaction wheel unloads, and more. These activities combined produce 4.7 million discrete resource changes throughout the simulation.

When new SPICE kernels are delivered, a 'Geometry' main method can be run that starts about a dozen geometry-calculating activities, some of which follow the 'Generator' pattern described in a previous section, and some of which make SPICE geometry finder calls. Which geometric quantities are calculated for which bodies, and what timesteps are used, is configurable via an input file. For the configuration values used currently, this simulation run takes about 8 minutes. Most of the computation time is spent inside the SPICE calls, which is difficult to reduce since SPICE is currently not thread-safe. However, this process only has to be run when kernels change, which is every few months typically, not every time any individual trade study has to be run.

Once the geometry TOL is computed, it can be read into the 'Mission Planning' main routine along with a list of a small number of manually placed activities. This main routine is the one run repeatedly for most trade studies, and its output is what populates the reference Mission Plan. After reading in the files, the code starts the decompose() methods of about 10 schedulers, which perform functions like determining the science observation vs. communication windows during each subphase of orbital ops, placing DSOC keep-out-zones, and determining instrument states.

After those, the DSN Track scheduler is run. The track scheduler itself and the interface to it are part of the multi-mission suite, and Psyche passes in 50 formal but human-readable rules that represent about 16 high-level ideas of when tracks should occur, examples of which could be "Continuous coverage using the spacecraft's HGA from between 3 and 7 days after launch", or "During Cruise phase, there should be 3 no-downlink thrust verification tracks every week." Complex track types can be appended together, as there are several cases where the mission planning team has determined it is desirable to have a telemetry track, a delta-DOR, and a DSOC opportunity all proximate to each other. However, the durations of the tracks do not necessarily need to be fixed, and view periods may not allow tracks to be completely adjacent, so the depth-first search scheduler tries varying the track plan recursively within the provided constraints until it finds one that meets the rules. If a track is over-constrained and cannot be placed, only it is omitted from the plan and the rest of the tracks are still scheduled. For Psyche, the process of placing 2,200 tracks in this manner takes fewer than 5 seconds.

Once the tracks and other activities which use data and power are placed, a REMODEL command is issued to the engine which produces resource timelines such as instrument states, data onboard per category, available array power, and more. The model() method for HGA Track references a custom telecommunications model through the multi-mission telecommunications interface and the multi-mission rate-based data model through the multi-mission data model interface. The activities that affect power also go through the multi-mission power interface, so while the power model is low-fidelity now, once a higher-fidelity one is available, none of the activity definitions need to change.

Once the modeling pass is over, the main method runs a few decomposition sections that place activities which do not affect resource values but help provide context and a more intuitive sense of the situation to the RAVEN visualization of the plan. Finally, in addition to the TOL, a data budget spreadsheet and scorecard that compares the DSOC opportunities provided against requirements are written out. The total scheduling, simulation and output together takes about one minute on a 2015 MacBook Pro. Prior to this capability, mission planners were using a mix of isolated C++, MATLAB scripts, and Excel which, while capable, were not designed to be used by a broader team or

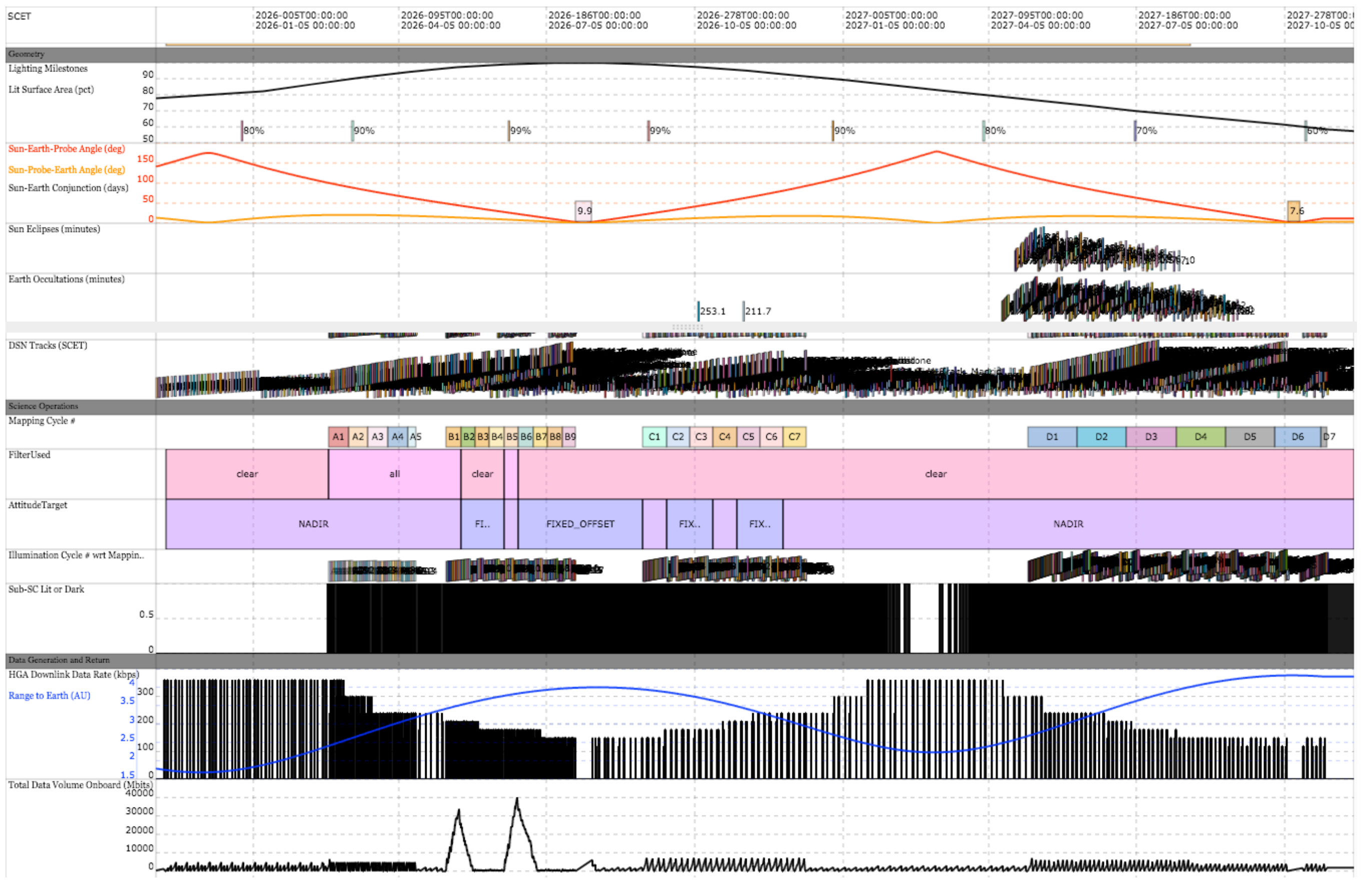

SCET
2026-005T00:00:00 2026-01-05 00:00:00
2026-095T00:00:00 2026-04-05 00:00:00
2026-186T00:00:00 2026-07-05 00:00:00
2026-278T00:00:00 2026-10-05 00:00:00
2027-005T00:00:00 2027-01-05 00:00:00
2027-095T00:00:00 2027-04-05 00:00:00
2027-186T00:00:00 2027-07-05 00:00:00
2027-278T00:00 2027-10-05 00
Geometry
Lighting Milestones
Lit Surface Area (pct)
80% 90% 99% 99% 90% 80% 70% 60%
Sun-Earth-Probe Angle (deg)
Sun-Probe-Earth Angle (deg)
Sun-Earth Conjunction (days)
9.9
7.6
Sun Eclipses (minutes)
Earth Occultations (minutes)
253.1
211.7
DSN Tracks (SCET)
Science Operations
Mapping Cycle #
A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 B6 B7 B8 B9 C1 C2 C3 C4 C5 C6 C7 D1 D2 D3 D4 D5 D6 D7
FilterUsed
clear
all
clear
clear
AttitudeTarget
NADIR
FI..
FIXED_OFFSET
FIX..
FIX..
NADIR
Illumination Cycle # wrt Mappin..
Sub-SC Lit or Dark
Data Generation and Return
HGA Downlink Data Rate (kbps)
Range to Earth (AU)
Total Data Volume Onboard (Mbits)

Figure 8. A screenshot of Psyche's reference Mission Plan Timeline, generated by Blackbird and displayed in RAVEN

maintained into operations. Before the Blackbird work began, doing simulation with a new set of assumptions required files being passed between multiple team members, so one could not rapidly iterate on a design. The current structure of the new system is presented in Figure 9.

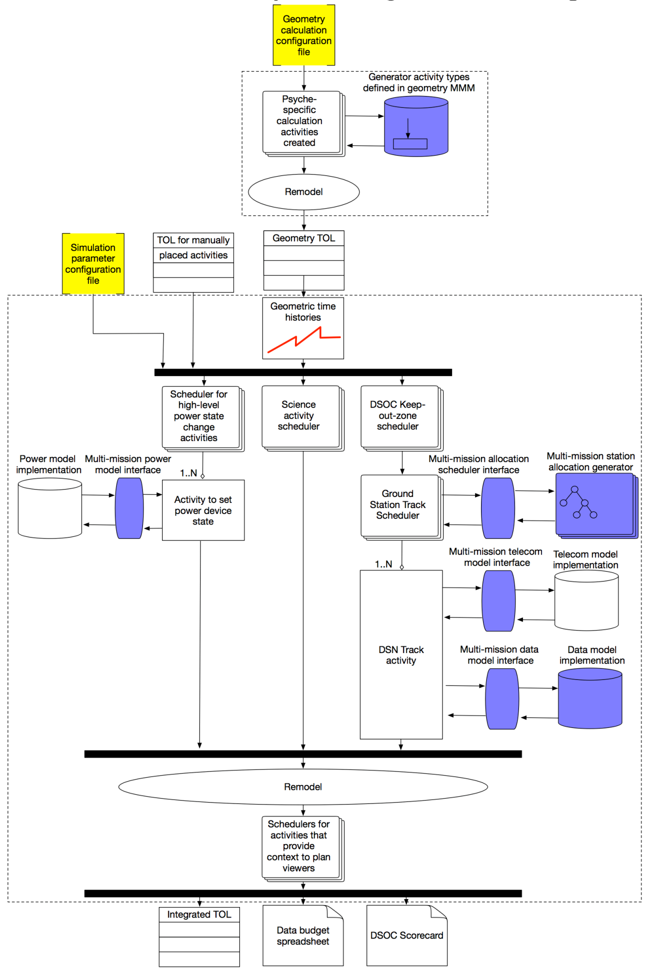

Geometry calculation configuration file
Generator activity types defined in geometry MMM
Psyche-specific calculation activities created
Remodel
Simulation parameter configuration file
TOL for manually placed activities
Geometry TOL
Geometric time histories
Scheduler for high-level power state change activities
Science activity scheduler
DSOC Keep-out-zone scheduler
Power model implementation
Multi-mission power model interface
1..N
Activity to set power device state
Multi-mission allocation scheduler interface
Multi-mission station allocation generator
Ground Station Track Scheduler
Multi-mission telecom model interface
Telecom model implementation
1..N
DSN Track activity
Multi-mission data model interface
Data model implementation
Remodel
Schedulers for activities that provide context to plan viewers
Integrated TOL
Data budget spreadsheet
DSOC Scorecard

Figure 9. High-level view of structure of and data flow through Psyche Blackbird adaptation

The mission planning team has used these new capabilities to support several trade studies, including the impact of regularizing observation sequences on the data budget, the feasibility of adding or removing DSN tracks during different phases of the mission, the sensitivity of orbital operation science activity timing to ephemeris uncertainty, the impact of using multiple data rates per pass instead of a single one, and more. The simulation output is also used to verify that there are sufficient DSOC opportunities to meet requirements and that the mission margin policies are being met.

Work is ongoing to use the output of the simulation to provide at-a-glance information about whether more requirements were met, like total time available for thrust. Blackbird is currently being used to help evaluate requirements for higher-fidelity subsystem models, and will be extended to communicate with those models once available.

Operations—A Blackbird adaptation is being built to perform several key roles in Phase E, including background sequence generation, thrust sequence generation, activity plan constraint checking, subsystem simulation, and serving as an interface to plan review tools. Psyche will be able to leverage activity definitions, constraints, and connections to other models set up for mission planning work for the operations activity planning work. As a proof of concept, one Psyche sequence engineer has already built the capability with Blackbird to generate valid Dawn background sequences for any set of Dawn input files. By leveraging the Blackbird multi-mission DSN model, this capability took less than 40 hours of work to implement. Psyche expects that using Blackbird will significantly decrease cost and risk over the alternative used for Dawn, which was a patchwork of Excel and standalone scripts that did not share code with other teams or missions. In large part due to how long it took these tools to generate products, Dawn's sequence generation and validation process took weeks per sequence, which ideally can be significantly decreased. Additionally, the Dawn toolset had to be frequently rewritten during different mission phases or when orbits changed, which can be avoided for the Psyche mission by appropriately parameterizing the schedulers.

InSight

InSight, JPL's most recent Mars surface mission, is dedicated to exploring the Martian interior. On InSight, the mission planner's primary responsibility is negotiating and scheduling Earth-Mars communication windows with other operating missions. This involves negotiating and scheduling 1) use of the DSN for Direct-to-Mars X-Band communication and 2) Ultra High Frequency (UHF) relay communication to the five current Mars orbiters. Using Martian orbiters as relay assets allows InSight to take advantage of the much more powerful antennas onboard the relay assets to transmit several orders of magnitude more data than is possible via Direct-to-Earth X-Band with the lander's own antenna. However, relay asset use must be negotiated amongst all the landed Mars missions. At landing, most mission planning tasks were completed manually in Excel workbooks. InSight has since taken advantage of Blackbird's powerful capabilities to automate mission planning tasks and generate command products.

The first task that was automated for mission planning was the optimization of Direct-to-Mars X-Band uplink windows. After DSN negotiations are complete, DSN schedulers provide mission planners information about DSN tracking windows. Within allocated DSN tracks, X-Band uplink windows should be scheduled for maximum uplink data volume and Radio Science Experiment (RISE) science return. Blackbird's simulation engine, built-in DSN model, and constraint-checking capabilities made it the best framework with which to automate this task.

The X-Band Blackbird scheduler works with three sets of input data: a set of Data Relay Capability Files (DRCF) that provide lists of windows during which uplink and downlink are supported at different rates, the SAF, and an Overflight Summary File (OSF) containing timing and parameters for all the orbiter relay overflights. For the DCRF, Blackbird's resource modeling capabilities track when uplink is supportable and at what rates. Blackbird's multi-mission DSN model reads the SAF and places Station Allocation activities automatically. Multi-mission code reads in the OSF and then a scheduler sets resources to indicate X-Band keep-out zones. The scheduler can then use resource-aware decomposition to avoid scheduling X-Band passes in parallel with UHF overflights. With these three sets of data, Blackbird calculates the best X-Band communication windows by matching DSN station allocations to the highest uplink rates that are supported within those allocations. This process takes hours in Excel and seconds with Blackbird.

The second task that was automated for mission planning was the creation of the Wake-Up Time Table (WUTT), a contingency command product that contains a set of wakeup, shutdown, and transmission commands. In the event that the lander stops executing its main onboard sequences, InSight will wake up and shut down according to the commands in the WUTT, and await instructions from Earth via the transmission commands. It would take a relatively serious anomaly for InSight to fall back to the WUTT, and therefore the transmission capabilities in the WUTT must be sufficient for rapid anomaly responses.

The WUTT must be power conservative in addition to containing sufficient transmission capabilities. There are scheduling constraints on wake-up entries, including: number of communication windows per day, maximizing data return and uplink volume, balancing X-Band and UHF transmissions, minimum and maximum sleep durations, total number of entries, and more. The WUTT can be generated using roughly the same sets of data that are used by the X-Band scheduler: 1) DRCFs to determine best X-Band times, 2), a SAF to determine available X-Band times and 3) an OSF from which to pick UHF relay overflights. From these inputs, the scheduler produces a WUTT that meets all of the requirements above, as well as a TOL and a review product. See Figure 10 for the structure of the InSight WUTT adaptation, and Figure 11 for the activities comprising a sample WUTT that Blackbird produced.

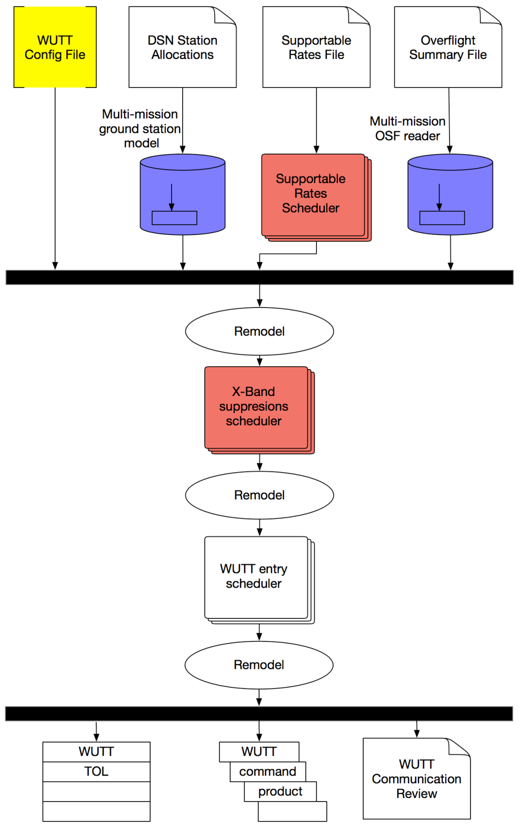


Figure 10. Structure of InSight WUTT adaptation. Red designates code shared with other mission planning routines.

The WUTT automation was validated with thorough regression and unit testing that was only possible due to the use of a standard programming language. These tests reduced the risk to the mission of switching to the automation and reduced the costs of verifying changes.

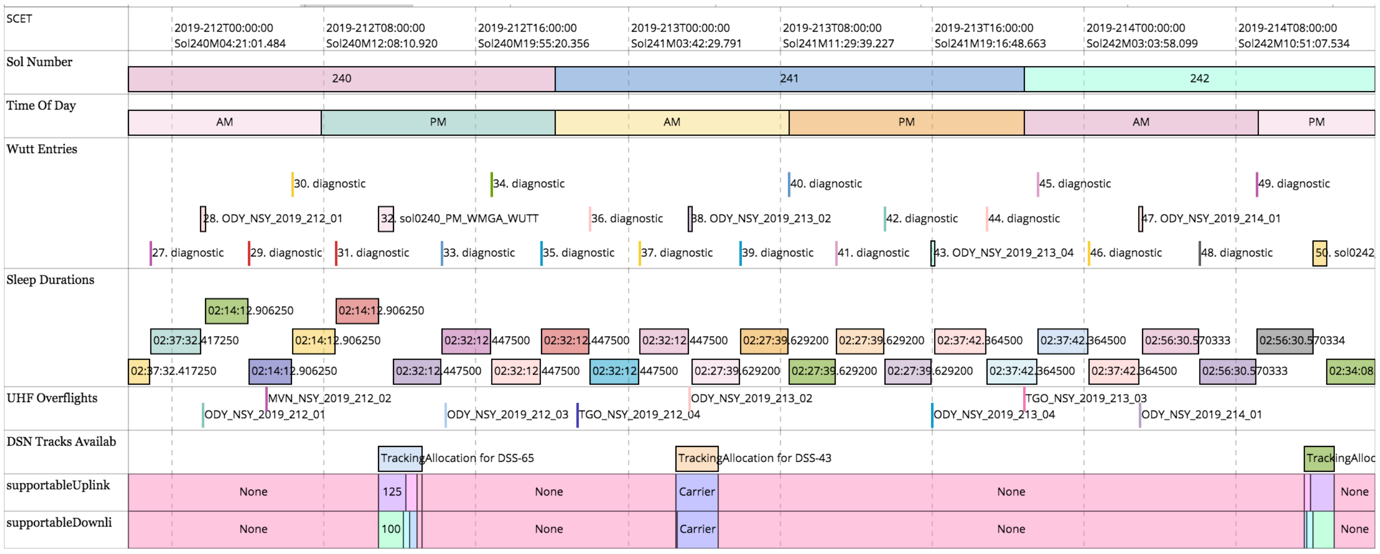

SCET
2019-212T00:00:00 Sol240M04:21:01.484
2019-212T08:00:00 Sol240M12:08:10.920
2019-212T16:00:00 Sol240M19:55:20.356
2019-213T00:00:00 Sol241M03:42:29.791
2019-213T08:00:00 Sol241M11:29:39.227
2019-213T16:00:00 Sol241M19:16:48.663
2019-214T00:00:00 Sol242M03:03:58.099
2019-214T08:00:00 Sol242M10:51:07.534
Sol Number
240
241
242
Time Of Day
AM
PM
Wutt Entries
Sleep Durations
UHF Overflights
ODY_NSY_2019_212_01
MVN_NSY_2019_212_02
ODY_NSY_2019_212_03
TGO_NSY_2019_212_04
ODY_NSY_2019_213_02
ODY_NSY_2019_213_04
TGO_NSY_2019_213_03
ODY_NSY_2019_214_01
DSN Tracks Availab
TrackingAllocation for DSS-65
TrackingAllocation for DSS-43
supportableUplink
supportableDownli
None
125
100
Carrier

Figure 11. Visualization of InSight's WUTT generation automation, as displayed in RAVEN

The third target task for mission planning automation is the creation of the Overflight Request File (ORF), which is upstream of the other two processes. In the manual process, mission planners must examine the list of geometrically available overflights, then deconflict with passes claimed by other landed assets to determine which to request, then perform calculations to determine which are the most viable. As currently envisioned, the automation will give the mission planner all the calculated information in one place they need to make the decision, then the passes will be manually chosen based on that.

NISAR

NISAR is an Earth-orbiting satellite whose mission is to use synthetic aperture radar (SAR) to map the Earth's surface every 12 days. After launch in 2022, the spacecraft will first undergo a commissioning phase that is required to be completed within 90 days before it begins science operations. During commissioning, operators need to deploy the spacecraft's antenna, perform checkouts of its maintenance systems, and perform more than 15 types of calibrations of the instruments [17]. Each type of calibration has prerequisites on the completion of other calibrations and engineering activities, and each require observations at different types of targets. In February 2019, the NISAR mission planning team began using Blackbird to help understand the time sensitivities of this highly interdependent system to variations in shift schedules, missed observation opportunities, activity durations, and other factors. Prior to starting to work with Blackbird, the creation of a mission plan that was self-consistent and satisfied the constraints was a manual process that took days, which made it challenging to react to changing calibration requirements and duration estimates.

Each calibration that must be scheduled consists of multiple instances each of five types of sub-activities: data acquisition, downlink, ancillary data delivery, data processing, and analysis. Data acquisition can only be scheduled during view periods provided by Systems Tool Kit, which can be for homogenous, point, or radar-bright ground targets, and it can only be scheduled when there is enough data storage and the relevant SAR is not being used for another acquisition. Downlink duration is modeled as proportional to the amount of data collected, and it can occur in parallel with waiting for ground reception of ancillary orbit products. Data processing takes a configurable constant amount of time per calibration type. Finally, analysis of the results for that observation can happen immediately afterwards for some calibration types, but for others the scheduler must wait until work hours for the operations personnel begin. Some, but not all, of the analyses can happen in parallel with each other or other components of the calibration. If an analysis must wait until another condition is true to begin, a nested forward dispatch scheduler is created to place it. After some calibrations have finished all their analysis segments, command preparation at JPL and uplink radiation at ISRO must be scheduled during their personnel's respective shifts, which are configurable. Some of these activities are visible in Figure 12.

In Blackbird, three different superclasses of calibrations were created to capture each underlying behavior: one that allowed parallel analyses, one that mandated serial analyses, and one that has one long analysis for all observations. For each calibration required, a line in an input configuration file specifies which of those behaviors is needed, whether uplink is required after the calibration is over, durations for the sub-activities, what target types are required, and the number of observations required. Each calibration becomes one activity with a Condition required for the calibration to begin, and all of them and their children's 75 total forward dispatch sections execute in parallel during a single remodel(), as shown in Figure 13.

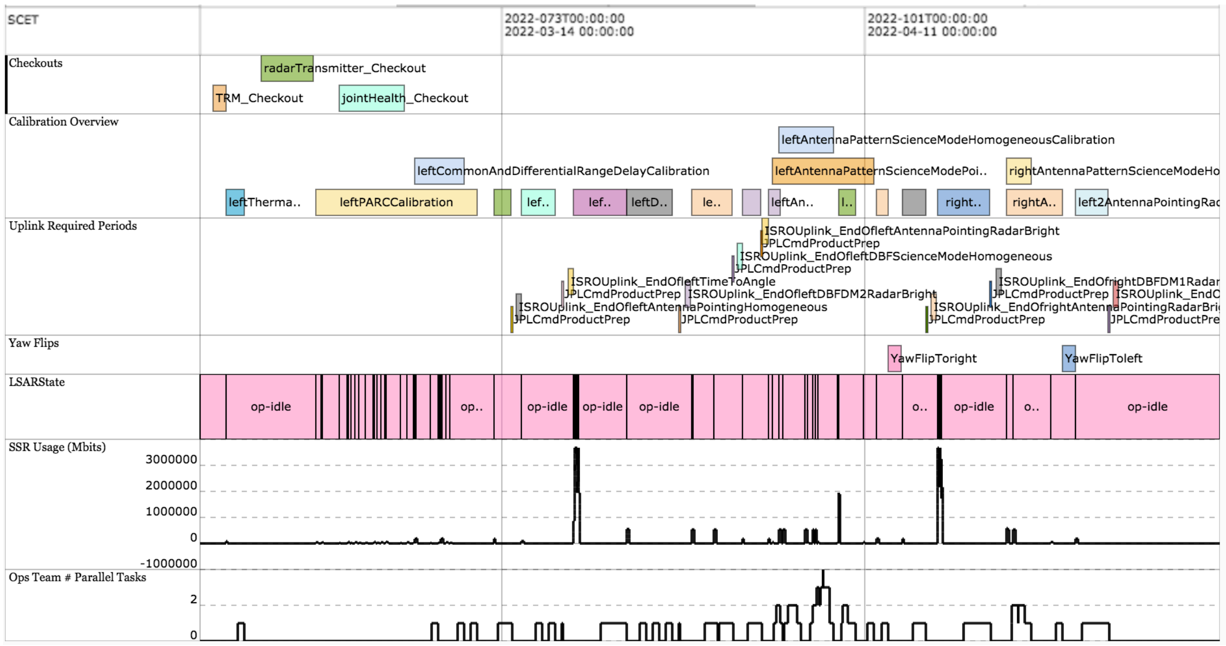


Figure 12. Output of NISAR's Blackbird adaptation, as displayed in RAVEN, showing the high-level activities

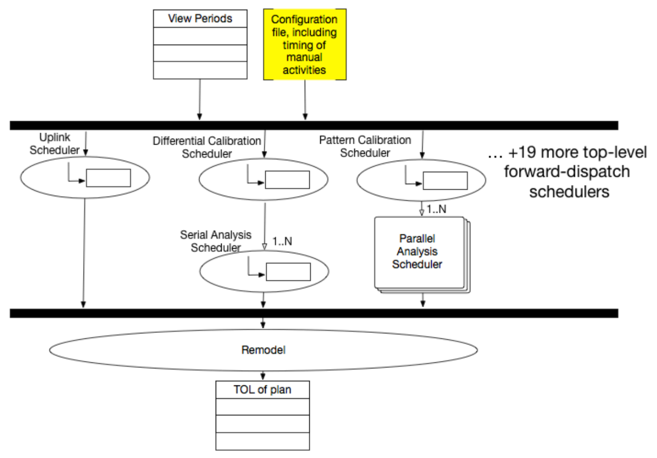

View Periods
Configuration file, including timing of manual activities
Uplink Scheduler
Differential Calibration Scheduler
Pattern Calibration Scheduler
... +19 more top-level forward-dispatch schedulers
1..N
Serial Analysis Scheduler
1..N
Parallel Analysis Scheduler
Remodel
TOL of plan

Figure 13. Structure of NISAR Blackbird adaptation

Blackbird's on-line scheduling capability makes expressing this problem relatively simple, even though NISAR was not one of the original mission use cases considered. Since the requirement is simply to minimize time spent in commissioning phase, a greedy approach is sufficient and no search is needed. That the engine automatically starts the threads when the conditions are available means the adapter can avoid having to manually work out algorithms that would place the sub-activities in a conflict-free way.

The adaptation was built by one engineer in less than two months half-time, and for the Mission System Critical Design Review, the project was able to use it to examine the effects of the aforementioned parameters on the timeline. Rescheduling the entire commissioning phase, about 1,000 constituent activities, for a given configuration file takes less than 10 seconds on a 2015 MacBook Pro. The simulation will continue to be refined going forward.

Europa Lander

Europa Lander is a mission concept that aims to sample the surface of Europa in the 2030s. The development team has been through several design iterations. For the most recent one, the concept was switched from a lander that would use a relay orbiter to one that would communicate Direct-to-Earth. Around the time of that change, it was decided to switch to Blackbird for future systems simulation work.

The Blackbird model takes as input a landing time and SPICE kernels describing the location of the lander, builds an activity plan, and simulates resources at the level of individual power devices and data channels to arrive at estimates for the lifespan of the Lander and its total science return. The activity plan is built using rules encoded into the scheduler based on geometric conditions. The scheduler decomposes into a tree of activities, as shown in Figure 14.

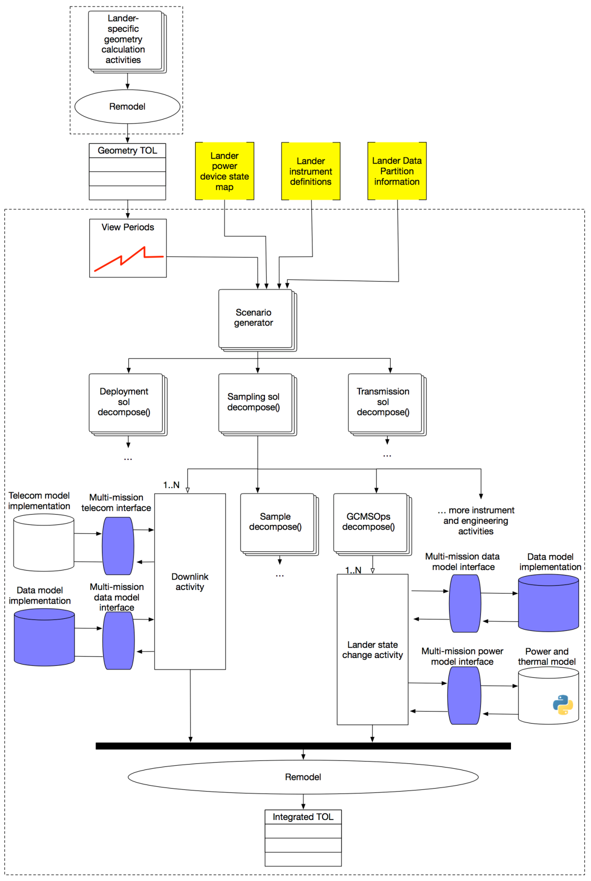

Lander-specific geometry calculation activities
Remodel
Geometry TOL
Lander power device state map
Lander instrument definitions
Lander Data Partition information
View Periods
Scenario generator
Deployment sol decompose()
Sampling sol decompose()
Transmission sol decompose()
...
...
1..N
Telecom model implementation
Multi-mission telecom interface
Downlink activity
Sample decompose()
GCMSOps decompose()
... more instrument and engineering activities
Data model implementation
Multi-mission data model interface
...
1..N
Multi-mission data model interface
Data model implementation
Lander state change activity
Multi-mission power model interface
Power and thermal model
Remodel
Integrated TOL

Figure 14. Abbreviated structure of and data flow through Lander Blackbird adaptation

The encapsulated nature of the activities means that large changes to the scenario can be created through changes to a single activity. For instance, modifying just the high-level Sampling Cycle activity can change which view periods are used for sampling, processing, or transmission. The adaptation heavily relies on the ability of activities to spawn other activities in complex combinations, which is greatly enabled by being able to write code to set behavior.

The Lander adaptation uses the multi-mission data model and the multi-mission power interface, which is implemented by a simple primary battery model that only tracks energy consumption. Since much of the energy could be used by component heaters, and because there are thermal constraints on the hardware that drive the mission timeline, a rudimentary thermal model also had to be included. Blackbird used inter-process communication to call the Mission Concept Review reference two-node thermal model written in Python and include those results in its timelines. The Blackbird adaptation is less than 1,000 lines of code and runs in 1.7 seconds when executing everything except the thermal model on a 2015 MacBook Pro.

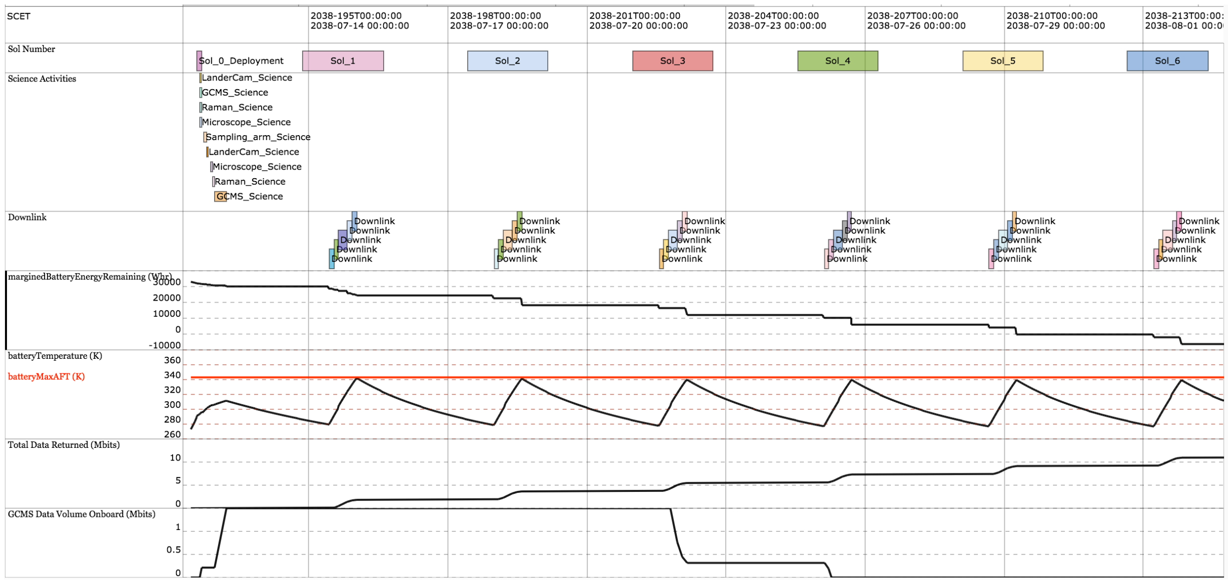


Figure 15. Subset of output of Europa Lander's Blackbird adaptation, as displayed in RAVEN

This adaptation is used primarily for trade studies. One of the more detailed studies involved examining the mission timeline in the event of an HGA failure which would necessitate using a low-gain antenna for downlink. The results of the study motivate future deeper exploration into antenna configuration and the possibility of using a medium-gain antenna for the lander. A sample plan created by the adaptation as displayed in RAVEN is shown in Figure 15, along with the resources predicted to be used.

Since Monte Carlo methods are of interest to the Lander team, the model was successfully run massively in parallel, where parameters for each run were chosen using Latin Hypercube Sampling. A python script sampled distributions to obtain parameters for each run and off kicked hundreds of Blackbird runs in parallel, each of which was deterministic given the parameters. The script then collated the data from the runs and created scatter plots showing quantities of interest like battery lifetime under different assumptions.

ASTERIA

The Arcsecond Space Telescope Enabling Research In Astrophysics (ASTERIA) is a 6U cubesat that was deployed from the International Space Station (ISS) into low Earth orbit in November 2017 for a three month prime mission [18]. Its prime mission involves imaging stars to detect orbiting exoplanets as a proof of concept to show that a cubesat can accomplish such a task. As of October 2019, the satellite is in its third extended mission and still detects orbiting exoplanets. In addition, the mission also hosts other experiments of various natures.

The Blackbird model for ASTERIA was developed because of a desire for more flexible and more automated planning tools. ASTERIA sequences are generated on the ground by tools written in MATLAB that partially validate the sequences. Typically, a sequencing engineer authors and validates daily background sequences for the spacecraft. The engineer must schedule the activities as well as check some flight rules manually. An approving mission manager must also double check these flight rules. Science sequences likewise have to be checked by a sequencing engineer and a manager. The current tools are too brittle to be adapted to automate these manual processes.

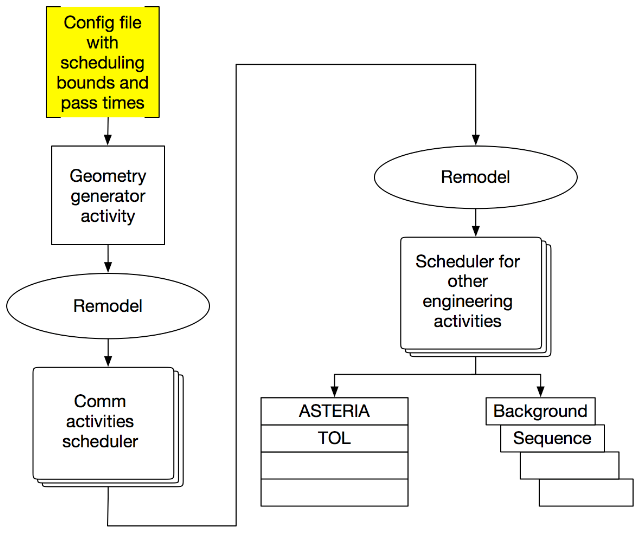

Config file with scheduling bounds and pass times
Geometry generator activity
Remodel
Comm activities scheduler
Remodel
Scheduler for other engineering activities
ASTERIA
TOL
Background
Sequence

Figure 16. Structure of current ASTERIA adaptation

In its current stage of development, the Blackbird adaptation is ready and being validated for use in operations. Blackbird produces a valid ASTERIA sequence in 5.1 seconds on a 2017 MacBook Pro. The adaptation is not fully integrated into the operations workflow and cannot yet be used to generate flight-ready sequences. However, there is a clear path forward to relieving the onus of scheduling background sequence activities for a sequencing engineer. Figure 16 displays the structure of the adaptation, and Figure 17 shows the activities which write entries in the output sequence.

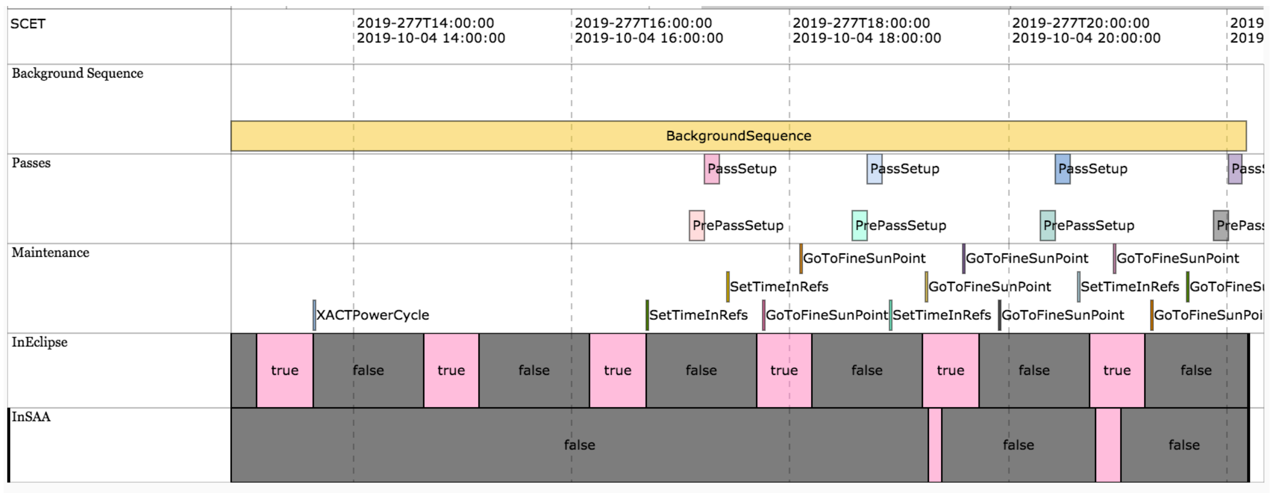


Figure 17. Activities scheduled by ASTERIA's Blackbird adaptation, as displayed in RAVEN

In order to generate the sequence, Blackbird's sequencing infrastructure had to be extended to write out the ASTERIA 'seq' format, which did not have prior support. Adding the classes needed to write out sequences in this format was completed start-to-finish in 2 hours. The entire rest of the adaptation to date has been built in less than one work-week by a single engineer who had little experience with Java before beginning this work.

Planned improvements to the adaptation will allow sequence engineers to author science observation sequences with a flexible GNC model. One of the most difficult constraints on science observation sequencing is avoiding a fault that trips when the angular momentum of the spacecraft is too high. This can occur when doing a prolonged observation at a single star, and will worsen as ASTERIA's orbit degrades. Better automation will allow the spacecraft to continue making full use of its capabilities, instead of losing observations with sub-optimal activity placement. By leveraging the modeling capabilities of Blackbird, the ASTERIA operations team expects to see increased science return with less effort needed from sequencing engineers.

6. Conclusions

Blackbird has proven itself as a tested operations-grade tool that reduces cost and risk on six missions currently. Designed with ease of use in mind, it has proved a natural choice for additional planning tasks beyond the traditional short-range activity planning. Its success is in large part due to the principles followed while in development, and specific design choices that shape how it behaves.

There is much work to do for each of the missions that use Blackbird, which is detailed in the corresponding subsections above. As that work occurs, a continued focus will be improving the multi-mission toolset to make it increasingly more capable and easier to use. In terms of the core framework, there are several ideas in work for improving usability, including increasing flexibility of the way activities can interact with conditions, and improving the ease of use for making complicated resources. The team is excited to work in this new open environment where engineers are encouraged to build on each other's work.

Acknowledgements

The work described by this paper was performed at JPL, managed by The California Institute of Technology, under contract to the National Aeronautics and Space Administration. The authors would like to thank Marcel Llopis, Daniel Finnerty, Debarati Chattopadhyay, Teerapat Khanampornpan, John Kwok, David Seal, Grace Tan-Wang, Pauline Hwang, and Priyanka Sharma for their support of Blackbird. The authors would also like to thank Zac McLaughlin and Samuel Fleischer for their early helpful feedback, and Sarah Bairstow, Marisol Arenas, and James Ashley for their detailed reviews of the paper.

References

[1] P. F. Maldague, S. S. Wissler, M. D. Lenda, and D. F. Finnerty, "APGEN Scheduling: 15 Years of Experience in Planning Automation," in *SpaceOps 2014 Conference*, Pasadena, CA, 2014.

[2] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, "ASPEN: A framework for automated planning and scheduling of spacecraft control and operations," in *Proc. International Symposium on AI, Robotics and Automation in Space*, 1997, pp. 181–187.

[3] F. L. Ridenhour, C. R. Lawler, K. Roffo, M. Smith, S. S. Wissler, and P. F. Maldague, "InSight Cruise and Surface Operations: Integrated Planning, Sequencing and Modeling using APGen," in *2018 SpaceOps Conference*, Marseille, France, 2018.

[4] M. D. Johnston and G. Miller, "Spike: Intelligent scheduling of hubble space telescope observations," *Intelligent Scheduling*, pp. 391–422, 1994.

[5] P. Van Der Plas, "MAPPS: a Science Planning tool supporting the ESA Solar System Missions," in *SpaceOps 2016 Conference*, Daejeon, Korea, 2016.

[6] G. Simonin, C. Artigues, E. Hebrard, and P. Lopez, "Scheduling Scientific Experiments on the Rosetta/Philae Mission," in *Principles and Practice of Constraint Programming*, vol. 7514, M. Milano, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 23–37.

[7] J. R. Doubleday, "Three petabytes or bust: planning science observations for NISAR," presented at the *SPIE Asia-Pacific Remote Sensing*, New Delhi, India, 2016, p. 988105.

[8] A. Van Deursen and P. Klint, "Little languages: Little maintenance?," *Journal of Software Maintenance: Research and Practice*, vol. 2, pp. 75-92, Mar. 1998.

[9] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, Dec. 2005.

[10] E. W. Ferguson, S. S. Wissler, B. K. Bradley, P. Maldague, J. Ludwinski, and C. R. Lawler, "Improving Spacecraft Design and Operability for Europa Clipper through High-Fidelity, Mission-Level Modeling and Simulation," in *2018 SpaceOps Conference*, Marseille, France, 2018.

[11] J. Gall, *Systemantics: the underground text of systems lore : how systems really work and especially how they fail*. Ann Arbor, MI (3200 W. Liberty, Ann Arbor 48103): General Systemantics Press, 1990.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley, 1994.

[13] "Projects," *The State of the Octoverse*. [Online]. Available: https://octoverse.github.com/projects.html. [Accessed: 02-Oct-2019].

[14] "TIOBE Index" [Online]. Available: https://www.tiobe.com/tiobe-index/. [Accessed: 02-Oct-2019].

[15] T. Suganuma et al., "Overview of the IBM Java Just-in-Time Compiler," *IBM Syst. J.*, vol. 39, no. 1, pp. 175–193, 2000.

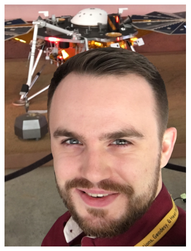
[16] C. A. Polanskey et al., "Psyche Science Operations Concept: Maximize Reuse to Minimize Risk," in *2018 SpaceOps Conference*, Marseille, France, 2018.

[17] P. Sharma, J. R. Doubleday, and S. Shaffer, "Instrument commissioning timeline for NASA-ISRO Synthetic Aperture Radar (NISAR)," in *2018 IEEE Aerospace Conference*, Big Sky, MT, 2018, pp. 1–13.

[18] A. Babuscia, P. Di Pasquale, M. W. Smith, and J. Taylor, "Arcsecond Space Telescope Enabling Research in Astrophysics (ASTERIA) Telecommunications," in *JPL DESCANSO Near Earth Design and Performance Summary Series*, Pasadena, CA, 2019.

BIOGRAPHY

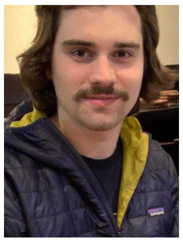
Christopher Lawler *received a B.S. in Mechanical Engineering with a minor in Computer Science from the University of Maryland, College Park. At JPL, he has worked on science planning, mission planning, and software development for InSight, Psyche, Europa Lander, Dawn, Europa Clipper, and NISAR. He is the lead developer of Blackbird.*



Forrest Ridenhour *received a B.S. in Aerospace Engineering from the University of Maryland, College Park. At JPL, he currently works on the Mars 2020 and InSight missions as a software developer for science planning, mission planning and sequencing. He is also a developer of Blackbird core. Previously, he has worked on the Juno, MRO, and Europa Clipper missions as a developer of activity planning and sequence validation software.*

Shaheer Khan *received a B.S. in Aerospace Engineering from the University of Maryland, College Park in 2017 and has since been with JPL. He has worked on the development of InSight's cruise and surface planning and sequencing software suites. He has also supported InSight during its launch, cruise, and surface operations in various roles. He is the current lead of InSight's Mission Planning and Sequencing Team.*

Nicholas Rossomando *received a B.S. in Aerospace Engineering from the University of Maryland, College Park in 2016 and has been at JPL since 2017. He works on sequence validation software and tactical operations design for the Mars 2020 rover mission. Previously, he worked on operations for the Opportunity Mars rover, including recovery efforts following the 2018 global dust storm.*



Ansel Rothstein-Dowden *received an M.S in Aerospace Engineering, a B.S. in Aerospace Engineering, and a B.S. in Applied Mathematics from the University of Colorado, Boulder. He is a sequence engineer on ASTERIA and works on planning software for Europa Clipper.*